MICE-0089-US

(99.03185)

APPLICATION

FOR

UNITED STATES LETTERS PATENT

TITLE: **COMMAND LINE OUTPUT REDIRECTION**

INVENTOR: **James McKeeth**

Express Mail No: **EL515088886US**

Date: November 24, 1999

Prepared by: TROP, PRUNER, HU & MILES, P.C.
HOUSTON, TEXAS
(VOICE) 713-468-8880 (FACSIMILE) 713-468-8883

COMMAND LINE OUTPUT REDIRECTION

Background

The invention relates generally to computer system support of application program execution and, more particularly but not by way of limitation, to a method and apparatus for redirecting command line utility output to a non-application maintained storage location.

Many applications such as word processing and file viewing programs have occasion to access system information. Often, such system information is available only through command line (e.g., console) utilities. That is, utilities that are accessible only through a command line interface. Illustrative command line utilities include "dir" and "net view" commands available in the Microsoft WINDOWS® operating system and the "w" command available in UNIX® and UNIX®-like operating systems (provides a list of users logged onto a specified computer system).

One difficulty with command line utilities is that their output is not generally directly useable by an executing application. The conventional technique by which a user application obtains command line utility output is shown in FIG. 1. After a temporary text file is created (block **100**), the command line utility whose output is desired is invoked via a standard interface (block **102**). Output from the command line utility is piped to the temporary file (block **104**), from which the application extracts and processes the desired data (block **106**). Sound programming practice calls for the destruction/removal of the temporary file created in block **100** (block **108**). It will be recognized that a temporary file may be created by the piping operation itself, i.e., during the acts of block **104**. Nevertheless, the use of a temporary file is generally considered essential.

A problem with the technique of FIG. 1 is that the application invoking the command line utility may not have file creation privileges on the computer system. If this is so, then the application will be unable to obtain the desired

data. Another problem is that if the disk the application has access to is full (i.e., incapable of accepting new or enlarged user files), any attempt to create a new file will generate an error. Yet another problem is that the file name chosen for the temporary file may already be in use. Still another problem is that many new PCs are disk-less and, thus, may not provide a mechanism through which user initiated (i.e., user invoked application) file input-output is possible. A further problem with prior art techniques such as that shown in FIG. 1 is that maintenance of temporary files is left to the calling application. If the application that creates a temporary file fails to remove it, a plethora of useless files may be generated over time.

Thus, it would be beneficial to provide a mechanism by which an application program may obtain output from a command line utility without the need to create a temporary file.

Summary

In one embodiment the invention provides a method to provide command line utility output to an application without the need of temporary files. The method includes receiving an identifier, receiving output from a command line utility, and storing the command line utility output in a system storage at a location identified by the identifier. In one illustrative embodiment, command line utility output is stored in a system registry database, which is stored in active memory (temporary storage) when the system is active and stored in a file (permanent storage) when the system is inactive. In another illustrative embodiment, command line utility output is stored in a shared system memory. The method may be stored in any media that is readable and executable by a computer system.

Brief Description of the Drawings

Figure 1 shows a prior art technique by which an application obtains command line utility output.

Figure 2 shows, in flowchart form, the operation of a redirection utility in accordance with one embodiment of the invention.

Figure 3 shows, in flowchart form, the operation of a one specific redirection utility in accordance with FIG. 2.

Figures 4 shows a block diagram of a computer system incorporating a redirection routine in accordance with FIG. 2.

Detailed Description

Referring to FIG. 2, redirection routine **200** in accordance with one embodiment of the invention uses a user/application specified identifier (block **202**) to identify command line utility output (block **204**) which it stores in a system-wide storage location (block **206**). By system-wide, it is meant that the storage location is available to all user applications and is, furthermore, maintained by operation of the underlying operating system. Following the act of storage in block **206**, a value associated with the identifier in the system storage is updated to indicate completion of the redirection routine and to, possibly, provide additional information to the calling application such as the amount (e.g., number of lines) of information stored. Once redirection routine **200** completes the act of storing in block **206**, the application invoking routine **200** may use the specified identifier to access the stored command line utility output.

One benefit of a redirection routine in accordance with FIG. 2 is that the calling application does not need file creation authority -- no temporary files are created. Another benefit is that there is no need for the calling application to remove temporary files as in prior art techniques such as that illustrated in FIG. 1. A corollary of this benefit is that the calling application does not require file deletion authorization. Yet another benefit of a redirection routine in accordance with the invention is that a second application cannot inadvertently destroy the results generated by a first application by accidentally replacing or deleting a temporary file (e.g., a background process designed to remove temporary files). Still another benefit of the invention is that the application invoking redirection

routine **200** does not have to have disk I/O (input-output) authority as the storage location is maintained by the underlying operating system -- the application makes I/O calls to the specified storage location through standard system calls (see discussion below).

By way of example, consider a situation in which an executing application needs information of the type provided by command line utility CMD-UTIL, where CMD-UTIL represents any utility executable from a command line prompt (e.g., the "dir" directory command of a Microsoft WINDOWS® operating system or the "head" command of a UNIX® operating system). In accordance with the invention, the application invokes a system call of the form:

CMD-UTIL [PARAM] | REDIRECT ID

Here, [PARAM] represents zero or more parameters that control or modify the execution of the CMD-UTIL utility, the "|" symbol represents the piping function available in many operating systems such as WINDOWS®, UNIX® and derivatives thereof, REDIRECT is the name of routine **200**, and ID is one or more parameters which REDIRECT routine **200** associates with output from CMD-UTIL during the act of storage in block **206** of FIG. 2.

It will be recognized that the calling application will generally ensure that the identifier it passes to routine **200** has either not been used or may be reused. It will further be recognized that command utilities may be stacked. That is, output from a first command utility (CMD-UTIL-1, for example) may be piped to a second, third, or Nth command utility (CMD-UTIL-N, for example) which may then be piped to routine **200**. In this case, a system call in accordance with the invention would be:

CMD-UTIL-1 [PARAM] | . . . | CMD-UTIL-N [PARAM] | REDIRECT ID,

where ". . ." represent one or more commands of the form CMD-UTIL-X [PARAM].

Because many current personal computer systems (PCs) are operated or controlled by one version or another of the Microsoft WINDOWS® operating system, an illustrative embodiment of redirection routine **200** utilizing the WINDOWS® system registry (hereinafter, the registry) will now be given. It will be recognized that the registry is an operating system generated and maintained database which application programs, application setup programs, and the operating system itself use to store configuration information.

Information stored in the registry is organized into hierarchical keys and associated key entries. Current versions of the registry use six predefined root keys (AKA Hives): HKEY__USERS; HKEY.CLASSES.ROOT; HKEY.CURRENT.USER; HKEY.CURRENT.CONFIG; HKEY_LOCAL.MACHINE; and HKEY.DYN.DATA. Each key in the registry can have one or more sub-key entries. Each key and sub-key can have one or more names (a unique character string identifier) and each name can have an associated value (data stored in a defined manor, may be a character string, binary data, a number, a Boolean value, etc.). Each key and sub-key has one default key entry that has no name.

Access to the registry is provided through system calls defined in the registry application programming interface (API). Illustrative registry API functions include: RegEnumKeyEx, which enumerates the sub-keys of a specified key; RegOpenKeyEx, which opens and returns a handle to a specified key; RegEnumValue, which enumerates the key entries associated with a specified key; RegQueryValueEx, which returns the assigned value of a specified key entry; RegSetValueEx, which assigns a value to a specified key entry, creating the key entry if the key entry was not previously registered; RegDeleteKey, which removes a key from the registry; and RegDeleteValue, which removes a key entry from the registry. Using keys (hereinafter understood to include sub-keys) and registry API system calls, routine **200** can store command line utility

output in the registry file. Using the same keys, an application program can retrieve information previously stored by routine **200**.

Referring now to FIG. 3, in one embodiment WINDOWS® based redirection routine **300** receives an identifier comprising a key from a calling application (block **302**). An illustrative key is HKEY.DYN.DATA/CMD-UTIL-OUTPUT-KEY. Routine **300** then begins receiving output from the CMD-UTIL utility, generally one line at a time as most command line utilities generate output targeted for line oriented standard output devices such as a computer display (block **304**). The received line is stored in the registry at a key name that uniquely identifies the line (block **306**). For example, each received line of output may be stored in the registry key:

HKEY. DYN. DATA/CMD-UTIL-OUTPUT-KEY,

with a name of "N," where "N" is set equal to 1 for the first received line, 2 for the second received line, and so forth. A test is then made to determine if additional command line utility output is available for storage (diamond **308**). If another line of output is available (the "yes" prong of diamond **308**), processing continues as block **304**. If no more output is available (the "no" prong of diamond **308**), the default value of the received key (i.e., HKEY_DYN_DATA/CMD-UTIL-OUTPUT-KEY) is set equal to a value corresponding to the total number of lines received and stored by routine **300** (block **310**). On completion, output from the command line utility CMD-UTIL is available for retrieval and manipulation by the calling application without the need to create, maintain or delete a temporary file.

In another embodiment, the ID parameter includes a storage location identifier. One value of the storage location identifier may direct use of the registry (or a similar operating system maintained database) while another value of the storage location identifier may direct use of operating system shared memory (e.g., volatile random access memory). One example of operating

system shared memory is the "clipboard" memory maintained by the WINDOWS® operating system.

Sub A7 Referring now to FIG. 4, illustrative computer system **400** in accordance with one embodiment of the invention includes redirection routine **400** (e.g., a routine in accordance with **200** and/or **300**) to redirect output from a command line utility to a specified operating system controlled memory location. As shown, routine **400** may be retained in storage device **404** which is coupled to processor **406** via system bus **408**. It will be understood that storage device **404** may represent non-volatile memory devices or a combination of volatile and non-volatile memory devices. Illustrative non-volatile storage devices include, but not limited to: semiconductor memory devices such as EPROM, EEPROM, and flash devices; magnetic disks (fixed, floppy, and removable); other magnetic media such as tape; and optical media such as CD-ROM disks. It will be further recognized that computer system **400** may incorporate one or more input-output devices **410** such as one or more secondary bus bridge circuits, memory controllers, accelerated graphics port devices and network interface adapters.

Various changes in the details of the illustrated operational methods as well as in the components of computer system **400** are possible without departing from the scope of the following claims. For instance, instructions to perform the acts of FIGS. 2 and 3 may be embodied in a program module implemented as, for example, a dynamic link library available through a standard interface. In addition, the illustrative system of FIG. 4 may include additional components such as network interface adapters and the like.

Thus, while the invention has been disclosed with respect to a limited number of embodiments, numerous modifications and variations will be appreciated by those skilled in the art. It is intended, therefore, that the following claims cover all such modifications and variations that may fall within the true sprit and scope of the invention.